Pablo's Guide to Estimation

Nothing is so useless as a general maxim.
(Lord Macaulay)

© 1998 Logica UK Limited.
Logica UK Limited's prior written consent is required before any part of this document is reproduced.

logica

Table Of Contents

1 Introduction 4
1.1 Estimidity and How it is Manifested 4
1.2 Why Estimate? 5
1.3 The Accuracy of Estimates 6
1.4 Who Should do the Estimates? 7
1.5 Can a Job Be Over-Estimated? 7
1.6 When Can An Estimate Be Provided? 8

2 Breaking the Problem Down 10
2.1 Familiarisation with the Project 10
2.2 The Estimatable Entities 10
2.3 The Spreadsheet 12
2.4 Tasks Easily Forgotten 17

3 The Estimation Process 19
3.1 Defining a Typical Task 19
3.2 The Moment of Truth 20
3.3 Assumptions 21
3.4 Reducing the List of Assumptions 23
3.5 Identifying Project-Wide Assumptions 23
3.6 Contingency 24
3.7 Separating Contingency and Task Estimates 25
3.8 Estimating Contingency 26
3.9 The Remaining Code/Test Tasks 27

4 Function Point Analysis and COCOMO 29
4.1 Science or Mysticism? 29
4.2 Do-It-Yourself Function Point Analysis 31

5 The Project Time-Scale 35
5.1 The Gantt Chart 35
5.2 Overhead Tasks 37
5.3 How Long Will the Project Last? 40

6 Project Roles 42

7 Project-Wide Contingency 46
7.1 Risk Analysis 46
7.2 Client Staff 47
7.3 Allowances for Reuse 47

8 Estimating the Project Cost 49

9 Reviews..... 51
9.1 Tidying Up 51
9.2 Types of Review..... 51
9.3 Now Do the Project 54

10 The Post Mortem..... 55

11 And Finally... 57
11.1 Further Information 59

1 Introduction

1.1 Estimidity and How it is Manifested

No-one wants to do the estimates. It is the most thankless task our industry can impose; a responsibility out of all proportion to the difficulty and risk of the work. An unsuccessful estimate can result in long hours for the project team, disruption to their lives and careers, sticky explanations to managers and clients and, above all, enormous financial loss. For example, if a Logica project loses a million pounds then it might take ten million pounds worth of successful work to regain the break-even point. Everybody remembers the name of the person who estimated a loss-making job but no-one ever can recall the one who estimated a project to the nearest day; it is just put down (usually with good reason) to the abilities and dedication of the team. If you want glory – don't do the estimates.

So there is an understandable reluctance for anybody ever to provide any estimates. There are many, often quite senior, people in the computing world today who will vacillate, delegate or even, at the last resort, flatly refuse to provide a decent estimate with their name appended. I call this attitude "estimidity". In this guide I shall discuss how estimidity is manifested and how it can be overcome. The aim is not to provide a 'method' by which reliable estimates can be obtained; indeed it is my belief that no such method exists or will ever exist. So you won't be able to appease your estimidity by saying "I followed Pablo's method and the thing was still a year late"; I may have overcome estimidity for my own projects but I'm not taking yours on as well. Instead of defining yet another methodology my aim is to make you feel good about your own estimates by ensuring that they really are your own and that they are the best that could be done at the time.

I have undertaken many estimates. And I have also spent time studying a single routine looking for some sign of why it took ten times longer to code up than its equally-sized brothers. Estimates are about experience. I do not just mean experience of other estimates, but experience of projects both successful and unsuccessful. I am reminded of a story about Gary Player, who once won a tournament with an amazing fifty-foot putt on the final hole. As they walked back to the clubhouse his partner remarked on Player's fortunate last shot. "Yes", replied Player, "It was a lucky one. But, you know, I find that the more I practice the luckier I get". Estimates are like that too.

Bad estimates lose us the good jobs and land us with the impossible ones. But they have to be done, and done to the best of our ability. So no more estimidity – let's go to the gallows at least informed if not actually laughing.

1.2 Why Estimate?

It is a curious fact that most software engineers assume that their profession is immune from the rules of other types of engineering and, indeed, of life itself.

I spend most summer weekends not far from Sizewell and have watched the progress of the new power station over the last few years (I hasten to add that this is not the main reason I spend time there; most of the area is still very beautiful). I couldn't help noticing that the methodology was not to dig a huge hole in the ground and then say "Hmm, what will we do now? Nuclear, gas or coal-fired? Pressurised water or gas-cooled?". Only *software* engineers work like that. "Let's build the easiest, cheapest, flashy, interesting, beneficial, cost-effective, quickest bit first, has come the cry from a thousand different projects, "And we'll worry about the rest later". And of course it is taken for granted that serious changes to functionality can be introduced at any stage, and all within the same timescale. Any other types of engineers would laugh at such requests; software people accept them as a fact of life.

This same unworldly approach is applied when considering estimation. The argument seems to run that accurate estimation won't get the job done any quicker, so let's just plough on until we're through. But how many times in work, in life in general, do we get this luxury? We use estimation every day: in calculating when to leave for work in order to get to our desk on time (but not *too* early), in deciding if there's space in the tube carriage for just one more body, in determining if there's enough Banana Bubbles left to last the week. It is just one factor in the thousands of judgements which govern our everyday lives.

Life, software and other major projects do not take place in isolation. To return to Sizewell, the engineers did not complete the reactor and *then* decide to order some fuel rods, which take three years to make. They didn't make the roof first and then start on the walls. (Okay, they actually *did* do that but I'm sure there was a good reason). Like computer projects, the work of different teams must be synchronised, hardware must arrive at the right time, users must be trained, cabling and air-conditioning needs must be met. And if each of these parallel strands is not estimated and controlled then the

timescale will start to stretch and the costs to rise. All too quickly the justification of undertaking the project in the first place have begun to disappear.

For projects are only worthwhile if they are cost effective and timely. If a new system saves £1 million a year but costs £30 million to develop then it is not going to start. If the software is to display the general election results or control a satellite, then it better be finished on time or the entire effort is wasted. I can think of very few cases in history where a major engineering development took place without any regard to costs or timescale. The pyramids? Stonehenge? The Apollo programme? Perhaps. But in most cases someone will want to know if commencing a project is justified. And they will continue to want to know this while the costs mount and the timescales lengthen. For there is always a point on any project where continuation is no longer worthwhile.

So at every stage of the project life-cycle from the vague idea to the post-mortem something is needed to provide a direction for the project and line management. And some sort of estimate, a plan, a timescale *does* make the job finish quicker. But an estimate is an aim, not a rigid timetable; a project and its staff still need to be managed and motivated if any sort of success is to be achieved. The estimates and plans should help, not be seen as a constant threat. Good projects, and good managers, are those who use such methods, not those who are slaves to them.

1.3 The Accuracy of Estimates

There are a number of “bloody obvious rules” defined in this paper. I have called them that because they are not rocket science, not precise results from years of scientific research, not even truths only someone as experienced as myself could have realised. They are rules we all know. And the first and most fundamental one is this:

> Pablo’s Bloody Obvious Rule of Estimation Number 1
>
> *Your estimate will be wrong.*

How can it be otherwise? You are being asked to predict the future. And it is not the future of something with so limited a count of contributory factors and end results as a horse race. It is a IT project, comprising any number of

events which will affect its progress, and probably change it out of all recognition. No-one can be expected to foresee what will happen to such a beast; and if anybody did then it can only be by sheer luck. Hell, if I could guarantee to undertake 100% correct predictions I would not be wasting my time on estimating computer projects – I'd take the easier option of the National Lottery results.

Of course ours is not the only business to be governed by wild guesswork, although in most organisations this is left to the management. Fortunately a sort of Parkinson's Law seems to apply to many projects. They eat the estimated cost, they eat the contingency, they eat the profit and then, almost exactly then, they stop. Maybe *this* is an argument against estimation; we could just set some arbitrary target and let the inevitable happen. But then how would the target be set?

1.4 Who Should do the Estimates?

This is one of the shortest sections in this paper because the answer is obvious. Estimates should be undertaken by those most qualified to do them and the best candidates are the people who are actually going to do the work. Clearly the estimators must have some sort of experience or they are going to have nothing to compare the new work with, so we are not talking about fresh-faced codewallahs straight from University. There can be no such thing as a 'professional estimator' because the experience of such a person would soon become so out-of date to be worthless. Even though I do a lot of estimation, I have to match that with actual project work or else by now I would be mapping mainframe/dumb terminal assumptions on to OO/Windows/SQL applications. Similarly, managers who have not personally undertaken any project work since dinosaurs ruled the earth should not be allowed anywhere near an estimate. So, the ideal estimator is a senior techie likely to work on the job.

1.5 Can a Job Be Over-Estimated?

Hopfstader's Law states: "Everything takes longer than you think it will, even if you take Hopfstader's Law into account". There's many projects which have seen the truth of that. But the downside is the belief that it is never wrong to over-estimate. This is seen at its most cynical in the widely quoted rule that "the person with the largest estimate is most likely to be right". Even if this rule is true it does not make for a positive attitude towards estimation. Over-estimates ensure that Logica fails to win a

competitive tender, or make a client decide that a project is not cost-effective after all.

It's no use claiming it's not your fault and that you just did some estimates and it wasn't you who decided not to go ahead. The fact is that although managers, clients and decision-makers may review and question your estimates, they still believe them; and if you have just gratuitously upgraded them and persistently stick to these inflated values just because of naked and unashamed estimidity then it really is your fault if new and interesting projects never seem to get started up.

But for developments that *do* start over-estimation is also a possibility. If you have seen a project where all the paperwork was done, a project where the staff had all their holiday when they wanted it, a project where no overtime was needed, a project where system testing showed up no faults, a project where a score of eight million was racked up on Tetris, or a project where every desk was free from listings, manuals, half-eaten burgers and polystyrene cups containing something Alexander Fleming would have given his right arm for, then you have seen an over-estimated project. In the heyday of computing (i.e. the early 70's) such things were the norm (you think all those Snoopy calendars were done in people's spare time?); today's world is more competitive and we can no longer afford to bid projects that will easily run to time and budget.

So we need precision in our estimates, not just a "it can't possibly take longer than that" attitude.

1.6 When Can An Estimate Be Provided?

I was passed in the corridor a few years ago by a salesman. "How long do you think it would take to do a C compiler?", he asked by way of greeting. I gave a number of man-years and we passed without a further word. I don't know why he wanted this number and probably never will (I have seen no sign of the job). The point is that *he didn't know* – possibly not even if it would take one or one thousand man years. He could have been on his way to a meeting where that lack of knowledge could have made him and our company look so inept as to be unemployable. Now, many techies would not have given that manager a straight answer to his question and I can see why (what machine? where? what version of C? what have we got to start with?). But that was not the requirement; the salesman wanted a number that minute. To have not given one would be estimidity in action. So we come to:

Pablo's Bloody Obvious Rule of Estimation Number 2

You can always provide an estimate.

I don't care if there are hundreds of things about the project that you don't know (what it does, what the performance requirements are, what the platform is, what methodology you're using, what the language is, what level of testing is needed...) it is always possible to come up with some number., regardless of the number of caveats you may care to set with it. Yes, I know that the number will be extracted, the caveats forgotten, and it will then be treated as is carved in stone. That is not your problem. Yet. But to chicken out of the task under a smokescreen of excuses is no help to anyone.

It is clear, however, that if we don't know a great deal about the project than the estimate is not going to be very good. And this leads us to:

Pablo's Bloody Obvious Rule of Estimation Number 3

The quality of your estimate depends on your familiarity with the task.

Because I believe in estimating entirely by 'feel' there is no short-cut here. I always want to know *everything* about the job: the history, the relationship with the client, the person who wrote the proposal, the quality of the specs, the performance requirements, even the names of the people currently off-charge who will be staffing the thing. The human brain works in a mysterious way. Try to think of the qualities that attract you about your friends, for example: the circumstances under which you met, what was said, how the relationship grew. You ended up with an accumulation of feelings achieved from a myriad of disparate pieces of information. At the time most of these facts seemed irrelevant, unrelated or even irrational but somehow they were synthesised to produce a 'policy' relevant to each friend. So when you are making your estimate nothing is unimportant; it all contributes to your overall feel about the project.

2 Breaking the Problem Down

2.1 Familiarisation with the Project

So, the first essential is to arm yourself with every relevant document you can, and to see all the people who are involved. Ask questions and get to know what the task you are being required to estimate is all about. Maybe it is something you are already familiar with, in which case this won't take long. But usually such familiarisation is the most important step in the estimation process.

However, you only have a limited time to do your estimate. Hell, I didn't do all this stuff when the manager asked me about the C compiler. But in normal circumstances I would maintain that at least 50% of the time available should be taken up with research and with the all-important period to allow your natural instinct for evaluation, abstraction and, yes, estimation to get to work. (I would like to allow more than 50% but I know how long it takes to assemble all the figures and charts for presentation.) You may well also be asked to come up with the ideal 'solution', the magic idea which will set our proposal apart from the competition, or get the all-important funding for your pet project. And this, you must admit, is not going to occur to you unless you have absorbed all the facts and set the subconscious to work.

2.2 The Estimatable Entities

So let's assume you have now got a good 'nose' for the project, have ascertained everything down the inside leg measurement of the Project Manager and are thirsting to begin your estimate. You now need to consider...

Pablo's Bloody Obvious Rule of Estimation Number 4

The more bits it's in, the longer it gets.

It is quite remarkable the way in which breaking a task down into smaller and smaller pieces causes an estimate to increase in size. Of course this is due to the fact that you are thinking about the task in more depth so more problems emerge and so more time needs to be added in. Furthermore there

is the ‘minimum size’ factor whereby no task can take less time than...what? one month? half a day? so the whole total is inflated. It was fortunate that I was not suffering from estimidity when I gave my estimate of compiler-size to the manager. Had I subsequently sat down and tried to do a design and then break that up into routines and then estimated all those I would undoubtedly have had to make an embarrassing call beginning “You remember that estimate...”.

So the rule is to break it into smaller bits. Whatever the stage of the project lifecycle at which you are having to do your estimate, you must be able to decompose the task into some manageable and more estimatable chunks. Obviously, there is a limit to the level to which you can go down, and this will depend on what you have to start with – you can’t start conceiving routine specs if all you have is a feasibility study. But what you do need is a list of functional, doable chunks, and I would say this should be 10 items at the very least, regardless of the size of the job in hand. If the project is in the concept stage, and we are following the normal Logica project lifecycle, then the chunks will be something like:

- Feasibility/Proof of Concept
- Functional Specification
- Prototype
- Technical Specification
- Code and Unit Test subsystems A, B, C...
- Integration Test Specification
- Integration Test (including time for fixes)
- System Test Specification
- System Test (including time for fixes)
- Acceptance Test Specification
- Acceptance Test (including time for fixes)
- User Documentation/Help Text
- User Training
- Cutover/Rollout
- Post Acceptance Support
- Project Management Overheads

If the project has progressed a bit and there is a list of functions (e.g. a Functional Specification has been written or can be extrapolated from the client’s requirements) then the Code and Unit Test elements can be broken down to a further level. This is the normal situation in my experience. Of the Technical Specification has been written then, of course, the

functionality can be broken down to the level of individual software components. More bits so a more accurate estimate.

We are thus in a position to follow:

> Pablo's Bloody Obvious Rule of Estimation Number 5
>
> *Draw up a list of tasks.*

The completeness of this list is vital. *Your estimate is far more likely to be inaccurate because you forget some piece of work, or some project role, than because you put an incorrect figure by each task.*

The list of tasks will also be useful in the review process. Logica standards demand two independent reviews before an estimate can be quoted. Agreeing the list of tasks with your fellow-estimator will allow the two sets of figures to be compared more easily. Also it is possible to agree the task list with other team members, people who have undertaken similar jobs in the past or even the client. Such people are usually far more willing to discuss the task list than to undertake any other part of the estimation process.

Your list of tasks should be entered into...

2.3 The Spreadsheet

Although one of the computer world's leading Luddites I do like to use a spreadsheet for estimates. Usually, the spreadsheet is in two parts: one for the code/unit test tasks and one for the overheads, such as project management, testing and specifications. My spreadsheets look something like this...

2.3.1 Code and Unit Test

Ref	Task	Days	Cont.	Total	Notes
A.1	Input of beers, wines and spirits	6	33%	8	Extra complexity on Fridays
A.2	Display signs of inebriation	2	10%	2	
A.3	Removal of traffic cone from head	3	100%	6	Unless surgery needed
:	:	:	:	:	:
:	:	:	:	:	:
:	:	:	:	:	:
	Total for Sub System A	**200**	**25%**	**250**	
:	:	:	:	:	:
	GRAND TOTAL	**1000**	**50%**	**1500**	

2.3.2 Overheads

Task	Days	Cont.	Total	Notes
Initial project lunch	6	33%	8	
End of lunch celebration	2	10%	2	
Recovery time	3	100%	6	
:	:	:	:	:
:	:	:	:	:
:	:	:	:	:
Overheads Total	2000	25%	2500	
Code and Test Total	1000	50%	1500	Carried forward
GRAND TOTAL	**3000**	**33%**	**4000**	
Plus Overall Contingency		10%	400	
FINAL TOTAL			**4400**	

The “Reference” column is usually a cross-reference to the requirements document or whatever you are working from. If there is nothing suitable then make up your own reference system or leave it out.

Sometimes I like to add a “Bucket” column which contains a code allocating each task to a particular class (e.g. “P1INIT”, “P2TEST”). This allows the eventual man-day totals to be broken down in several different ways (e.g. by phase, by subsystem, by stage in the lifecycle) which is required by some ITTs. (Use the SUMIF instruction in Excel for this).

The “Notes” column usually contains remarks about what is included or not included in this line or any particular reason why the number of days or contingency is unusually high or low.

“Cont” stands for contingency – I will be discussing this later. For the present just note that it is a percentage added on to the “Days” column to give the “Total” column. Round to the nearest day (not up or down). The contingency in the “Grand Total” row is the overall figure; in the example this is 1000 days of contingency expressed as a percentage on the ‘base’ figure of 3000 days i.e. 33%. This can also be calculated by the spreadsheet.

The other figures, I’m afraid, you have to work out for yourself.

2.3.3 A Genuine Example

Here are a couple of pages from a recent estimate I undertook.

Bucket	Task	Days	Cont.	Days	Comments
	PHASE 1A				Implement Point of Sale Facilities
	Code & Unit Test				
1I	SMART Interface	15	40%	21	File handoff every 15 mins.
1I	SMART Adapter	8	50%	12	Converts fields to internal formats, rules
1I	Record Trade	12	15%	14	No validation, Generic Format.
1I	Cancel/Amend	12	40%	17	Doubtless problematic
1I	Validate Trade	10	15%	12	
1I	Fix Validation Failures	10	15%	12	Screen(s)
1I	Basic Enrichment	4	15%	5	Only what is needed for POS Facilities. Probably not RRD so need to set up our own database or access another
1I	Fix Enrichment Failures	4	15%	5	Screen. Ditto. Possibly redundant.
1I	Access External Reference Data	20	50%	30	Only what is needed for POS and CSP. May be RRD Or another database or something we set up ourselves. Possible comms problems.
1I	Trade Enquiry	12	25%	15	Screen. Almost the final thing. Equities only, status limited. Problems with REPOs?
1I	Enter Splits	8	15%	9	Screen. Will feed final splits into 'Record Trade'. Manual entry only (anthing from ETC must be entered manually)
1I	Display 'Accumulated' Trades	6	15%	7	Screen. A form of enquiry
1I	Logon/Off	2	15%	2	Screen. Internal static data for users/privileges
1I	Maintain Static Data	8	15%	9	Screens. Includes users.
1I	Settlement Interface	12	50%	18	To CSP. Assume file handoff. May require additional 'enrichment' data from RRD or wherever.
1I	Fix Settlement Problems	4	15%	5	Screen. Fixes to 'enriching' CSP data. Possibly redundant.
1I	Permissioning mechanism	10	15%	12	Access by function (greying), some fields read-only etc. Establish mechanism for entire application.
1I	Audit Trail	0	15%	0	Assume part of the functionality above. Trades carry history of events plus user name if manually adjusted. Can examine audit trail by one-off enquiries, not budgeted for here.
1I	Operator Display	5	15%	6	Screen. BSF status
	C&UT Total			**211**	

	Overheads				
1M	Initial Factfinding	10	15%	12	
1M	BSF Infrastructure	80	25%	100	Making the infrastructure 'production quality'.Plus known improvements
1M	BSF Maintenance	0	40%	0	Included in above
1I	Functional Specifications	20	15%	23	As well as Functional Authority (full time)
1I	Prototype	4	15%	5	Screens. Assume final screens carried forward to implementation
1I	Agree FS/Prototype	4	25%	5	
1I	Technical Designs	35	30%	46	As well as technical architect (full time)
1I	Database Design	100	15%	115	Physical and Logical models. Full time role
1M	Team briefings	70	0%	70	
1M	Development System setup	15	30%	20	
1I	System Management (P1)	60	20%	72	Includes Forte Repository management. This will be a full-time role as the project expands and complete familiarisation is important, hence assume full-time from the start.
2I	System Management (P2)	40	20%	48	
1I	Forte Guru (P1)	15	40%	21	
2I	Forte Guru (P2)	5	40%	7	
1I	Simulators, Test Tools	15	30%	20	SMART simulator, Compare settlement records with existing system?
1I	Test Data Maintenance	0	0%	0	Job of database man
1D	External Data	0	0%	0	Support is job of databse man, otherwise out of scope.
1D	Data Take-On	0	0%	0	Support is job of databse man, otherwise out of scope.
1I	Project Director (P1)	60	15%	69	
2I	Project Director (P2)	40	15%	46	
1I	Dev Project Management (P1)	60	15%	69	
2I	Dev Project Management (P2)	40	15%	46	
1I	Functional Authority (P1)	60	15%	69	
2I	Functional Authority (P2)	40	15%	46	
1I	Technical Architect (P1)	60	15%	69	
2I	Technical Architect (P2)	40	15%	46	
1I	Team Leading (P1)	0	0%	0	Assume senior staff for this phase.
2I	Team Leading (P2)	0	0%	0	
1I	Team Overhead (P1)	0	15%	0	Walkthroughs, Reviews etc. Assume senior staff allows overhead to be absorbed in C&UT time.
2I	Team Overhead (P2)	0	15%	0	
1I	Manuals, Help text	2	15%	2	Some help text in this phase. No manual.
1D	Failover, performance	20	30%	26	Support, testing
1I	System/Integration Test Specs	25	15%	29	
1I	System/Integration Test	80	20%	96	Includes fixes by dev team.
1D	Parallel running	20	30%	26	Support
1D	Acceptance Test Spec	25	15%	29	
1D	Acceptance Test	35	30%	46	Including bug fixes
1D	User training	10	15%	12	Not much to demo, although quite a few users.
1D	Live system support	25	30%	33	
	Overheads Total			**1323**	
	Phase 1A Total			**1534**	

2.4 Tasks Easily Forgotten

As I said, the omission of tasks is more critical than the estimation of each one. Some which are regularly ignored are:

- Familiarisation (see below).
- Time it takes to get documents agreed with the client or users (often causing delay).
- Design and agreement of the user interface.
- Integration (simply building a large system can be a slow process every day).
- Processing of change requests, and the effect of these on the developers.
- Test data set-up and maintenance.
- Operator facilities to monitor and run the system.
- Testing of fallback procedures.
- Help text on user screens.
- Setting up and understanding bought-in packages.
- Testing that performance and sizing requirements are met.
- Setting up the development environment.
- Moving from the development environment to the target.
- Correcting problems found during testing.
- Bits of code to be written that are not directly attributable to any one function. Things like:
  - common routines
  - interfaces to hardware
  - interfaces to ‘third party’ software packages
  - testing/tracing tools, harnesses, simulators
  - system start-up/shutdown
  - backups, archiving
  - ‘programs’ in SQL, macros, shell scripts.

2.4.1 Familiarisation

By familiarisation I mean, for each team-member:

- Getting familiar with the project as a whole.
- Getting familiar with the bit of the project that they are responsible for.
- Finding out about the project environment (e.g. what exactly they have to do to compile something).

- Learning to use any tools or methods with which they are unfamiliar (not formal training, but actual hands-on use for this particular project).

All techies know that these tasks are significant at the start of a project and should be allowed for. If they are not then the project tends to find itself behind schedule within the first month

Despite this, I have always found it difficult to get an allowance for familiarisation through a management review of an estimate. “I think”, they say, striking a firm line through the spreadsheet, “that we can always get suitably skilled staff. And if not, then we will train them up as part of the company overhead”. But it never happens. The staff arrive direct from a spell on-site in Bromsgrove, thinking in terms of a good rest, and are immediately confronted with something completely strange to code up. The training never happens or is delayed so long that the staff know more than the trainers.

I’ve tried using every buzz-word I know – ‘training’, ‘ramp-up’, ‘familiarisation’, ‘learning curve’, ‘briefings’ – but it’s no use. No manager seems willing to admit that their staff are unable to start coding on a new project at the drop of a hat. The only alternatives are to add this ridiculous assumption to the list, or secretly to weight the estimate somewhere unobtrusive.

The situation is even worse when the client is able to see the task breakdown. “Why should Logica staff get trained up at my expense?”, is often their argument. Even though we all know that the problem would still be there even if the project were staffed by the client themselves, the pressure is often to remove or hide this overhead.

3 The Estimation Process

3.1 Defining a Typical Task

Usually, we will want to estimate the code/unit test tasks first. This will either allow us to estimate the overhead tasks on a *pro rata* basis or at least give us some sort of idea that our estimate for these is correct.

It is often the case that the code tasks are all of roughly the same type, or that there are a number of groups of tasks of the same type. For example most of the code might be a "get some screen input, do some database access, do some calculations, put the result on the screen" type, or a "get the report parameters, access the database, print the report lines, calculate and print the totals" type.

If tasks can be grouped like this then the best approach is to select a typical task, spend some time in estimating this and then determine the rest *pro rata*. This is further discussed below. If the task set is completely disparate then you will have to undertake a unique estimate for each one. Hard luck.

But usually we can follow...

Pablo's Bloody Obvious Rule of Estimation Number 6
Identify and estimate a baseline task first.

Select a task or a function, any one from your list, that seems to you to be typically simple. If the list contains a number of different types of task then choose one of each type. Well, of course, "typically simple" is a relative term and very dependent on the job you have in hand and the stage in the project at which the estimate is made. How can you choose?

Let's take an example. We are at the proposal stage in the project and there is a requirements spec to work from. We know that the solution requires a database and that there are transactions to set up data, do enquiries and produce reports. For our baseline task we are looking for a run-of-the-mill transaction which, we'll say *for this particular project*, calls for a couple of database entities to be read, a few lines of calculation to be done, the results to be displayed, screen input accepted and validated and a couple of

database entities to be written. Similarly we might pick a smallish report that needs a few entities to be read, with a bit of calculation and formatting to be done. So two types of task and two "typically simple" functions.

"A couple", "a few lines"; I know it all sounds pretty vague. But this is where your knowledge of the job as a whole comes in. Remember that you are looking for are "typically simple" tasks in terms of your particular project. Maybe a typical transaction would call for one entity to be read, maybe six. What is the typical amount of "calculation" needed? Maybe two lines of code, maybe ten pages. Maybe your system is nothing like this – process control, perhaps, where you are looking for a typical task in terms of the number of inputs and outputs and the complexity of the interrelationships between them. I can't help you choose; but what I do know is that nestling in your list of functions or tasks are one or more which are "typically simple" and that these will be the baseline, the cornerstone of your estimate.

3.2 The Moment of Truth

In theory, the full code/unit test process consists of the following stages:

- Understand the function and interfaces to the code unit, using the Technical Specification, with help from the Team Leader.
- Write the code.
- Write a Unit Test Specification.
- Write any test harness needed to undertake the unit test.
- Submit code for review, and undertake changes required.
- Submit Unit Test Specification for approval, and undertake changes required.
- Undertake the unit test, correcting faults which arise, and re-test.
- Submit the completed module for integration testing.

This may seem a lot, and you will have to use your judgement as to how much of it you expect to happen in practice, but remember that a day is quite a long time and that some of these processes are a matter of a few minutes.

So now you can estimate the time it will take to code/test the baseline task.

At this point the readership will divide into two groups: those who read the last sentence and plucked a number from the air, and those who could not. We will return to the first group shortly, but let us examine the second.

There are two possible reasons why you can't come up with a figure: either you have a terminal attack of estimidity or else there are too many unknowns. The first case I cannot help other than to point out that you have to do this estimate sometime and it's no good just gibbering at it. Come on now, let's at least find a range of possible numbers.

Could the task be done in a day? No way.

A day? Never.
A week? Well if a miracle happened.
Six weeks? Well, unless something went badly wrong.

And now from the other direction.

Could the task possibly take five months? Surely only if disaster struck.
One month? Very difficult to see how...
Six weeks? Aha!

3.3 Assumptions

Okay let us assume that you now have a figure, or at least a range. But whether you guessed a figure in an instant or struggled for three days, you will inevitably have made a lot of assumptions. 'Feel' is all very well but...

a) You are going to be asked to justify those numbers sometime.

b) You will probably change them once you start thinking about the assumptions.

c) You will need to estimate the contingency in a minute.

So the hard bit now is to apply...

Pablo's Bloody Obvious Rule of Estimation Number 7

Write down your assumptions.

Now this *is* tricky. It's this old problem about the mysterious workings of the brain again. Can we really ever know what subconscious assumptions we are making? Well, no. But we can at least make a list of the conscious ones. So don't forget here things like:

- Quality and depth of the documents you have seen (the stability of the requirements).
- Quality of potential/available staff and prerequisite knowledge.
- Availability of tools and support software.
- Dependence on bought-in packages.
- Level of testing needed (e.g. will we need to do *written* unit test plans/results?)
- Development environment (technical and site considerations).
- Availability of knowledgeable people.
- Size of project and possible time-pressures.
- How much interference you expect, number of changes that may be needed.
- Any constraints on reliability, storage, performance, quality etc.
- 'Must meet' dates.

Your list will be very different from this, longer and *unique to your project*.

Make your list and see how hot that figure you plucked from the air, or wrestled over, still feels.

Remember that the aim is to make you feel good about your estimates; and one way to do that is to know what it is that you are worried about. But don't cheat. Don't put down such a great long list of assumptions that it's almost certain that some will turn out to be untrue. This is another manifestation of estimidity. I don't want to see something like "Assumes that the staff are all very competent" in your list; you know what the average is, you may even know who the proposed staff are, so your estimate must embrace that.

3.4 Reducing the List of Assumptions

Of course you must do all you can to reduce your list of assumptions. Some are and will always remain unknowns but anything which you can establish as a fact will help. Eventually your list of assumptions will be accompanying your estimate. It is an essential part of it. Indeed some of these assumptions may be placed into a proposal. However, others may show you, your co-workers or your company in too bad a light to make explicit. You can't put "we assume that we'll be able to find someone who has used SQL before" in a proposal. It might be one of your assumptions but let's keep it to ourselves.

But there are some you *can* do something about. Don't you know enough about the hardware platform? Well, ask. Ask the client, get help from other people in the company, get some manufacturer's literature if there's time. Is it important that a Project Manager is experienced with this size of project? Get some names of possible candidates. Again the rule is to split things up as much as possible. If one assumption is for some skilled technical help from the client then ask yourself which functions this will be needed for and what skills are needed at each stage.

3.5 Identifying Project-Wide Assumptions

Some of your assumptions, such as "assumes that message handling will be provided by the TP Monitor" might be considered *project-wide* while others, such as "assumes this module can reuse some code from the last one" may be *specific to a particular task*. We need to separate these two groups so we don't have to build the project-wide assumptions into the contingency for each code and unit test task. As we will see, it is easier to cater for these at the end.

For example, let us say that you don't know if the user interface for your system is to control the screen directly or to use a package that will make the user interface relatively 'easy'. The 'raw' estimate for your baseline task assumes the package is available. If the assumption is false and there is no suitable package then there is a separate task to write your own. This is project-wide estimatable task in its own right. It may become zero days if the 'package' route is eventually chosen., but in the worst case at least we can estimate a figure for the other eventuality.

The net result should be that you have a list of project-wide assumptions and a list of those which are specific only to your baseline task. Your project-

wide assumptions are *risks*, and will form a part of the project-wide contingency that I will discuss later.

Contingency is intimately connected with our lists of assumptions, but first let me digress and discuss the concept of contingency in general.

3.6 Contingency

The word “contingency” is often misused. Managers affect to presume it is something that techies build in to their estimates in order to ensure an easy time during the development. Worse still they assume it is in some way synonymous with, or additional to, the profit made by a project. It is not. It is your estimate of the percentage error on your ‘raw’ estimate.

The important things to note are firstly that the contingency is another estimate, and secondly that it must always accompany any ‘raw’ figure. It is a little like the way in which resistors are always quoted as being, say 100Ω with a 1% tolerance. The difference with contingency is that it is always on the upwards side; the resistor may be 99Ω but a task estimated at 20 days with 50% contingency is never going to take 10 days. What this pair of figures says is that even if all our assumptions are incorrect then it is unlikely to take more than 30 days to do the task.

At the very least you ought to be able to guess a “couldn’t possibly be more than that” sort of figure. But I am not saying that this is the one you should use. It is the ‘with contingency’ figures are the ones you are going to be using from now on and which will make up your final estimate. In a reasonable world some of the assumptions may work out for the worst while others may just disappear. The idea is not to take the gloomiest possible view but a considered risk. You can never cater for every possible situation or else the contingency will always be infinite. So, to go back to our example, we are not saying it would be *impossible* for the task to take longer than 30 days. All we are saying is that this is a reasonable maximum to allow at this stage.

There are actually two sorts of contingency, and it is important not to get these confused. One sort is dependent on the assumptions specific to each task. It represents how confident you are about the number of days you have allocated *to that particular task*. The other sort is the project-wide contingency, which represents the risks applicable *to the estimate as a whole*. This is dependent on the project-wide assumptions. In either case...

Pablo's Bloody Obvious Rule of Estimation Number 8

The contingency is the sum of the risks estimated for each assumption.

So our project-wide contingency is depenent on the project-wide assumpriosn which we have identified and the contingency for each task depends on the assumptions particular to it.

All we have to do is estimate the amount of risk (as a percentage increase) associated with each assumption, add it up and there is our contingency figure. Easy. But these percentages mount up; in no time you seem to have 300% contingency on every task. You must try to get as many of the assumptions as possible resolved in some way because this will increase the validity and credibility of your estimate.

3.7 Separating Contingency and Task Estimates

Earlier, you may have made a sneaky upgrade to your estimate having drawn up your list of assumptions. That's okay so long as you really think the job *ought* to take that long. What is not okay is if you think it *might* take that long. Mixing up the estimate with the contingency is a very common habit. and the reason is very simple...

Pablo's Bloody Obvious Rule of Estimation Number 9

It's harder to estimate the contingency than the task

Why? Well, you are an experienced sort of person, which is why you were given this estimate to do in the first place. You know that, all being well, this typical task would take you X days. Max. But it's probably not going to be you and, anyway, what if...? what if...? what if... ? That's the hard bit.

But everybody falls into the trap of bumping up the base estimate rather than the contingency. Why? Well, managers are partly to blame; they can't

understand why all the contingency levels are so high and arbitrarily axe them. Contingency, they nobly say, chins out and mouths hardening to a determined line, while the *Dambusters* theme swells in the background, is theirs to bear. Lies, all lies. The estimate is still yours and if the things you thought would go wrong do go wrong (along with the things you never thought of) who do you think will end up getting blamed? But the way out is not to just inflate the base estimates to cater for the contingency you know is going to be lopped off. This is your estimate, remember and the aim is to keep you happy with it. So present it straight. If, however, someone can set your mind at rest about the assumptions you have made then it is quite valid to reduce the contingency.

Apart from this problem, some estimators think they are in some way at fault if the contingency goes above a particular figure – 100% is often quoted. After all, what sort of a crummy estimate are you doing if you can't tell if a typical task will take three months or six? So the temptation again is to inflate the base figure, which tends to remain inviolate during reviews and have a less frightening contingency figure by it.

But be brave. If the situation is one where the unknowns are great and the risk is high then management ought to know about it. So hang on to your list of assumptions and the contingency you have applied; these will be accompanying your baseline estimate.

3.8 Estimating Contingency

To estimate the contingency, you are going to have to put a percentage risk figure by each of your task-specific assumptions. (We will be covering the project-wide assumptions later.) There is no magic formula for this. I may look at a task and say "well, I don't really know the algorithm by which the number of input widgets becomes translated into the monthly production target of truss flanges. It can't really be too tricky because I phoned the guy who does it by hand at present and he says that there's just a couple of special cases at the end of the month. Let's say it ought to take 20 days. Assume that these special cases aren't *too* difficult – say 5 days or 25%".

You see what I mean about the estimation of contingency being more difficult than the task.

Look back at my "genuine example" earlier and you will see that I have placed 15% contingency on nearly all tasks. This reflects a standard level of doubt I had at the stage I undertook the estimate. The 15% could have been

assigned as project-wide contingency if I had wanted, but would have disadvantages if I later wanted to reduce or remove the contingency for a particular task. Where my contingency estimate rose above 15% I have noted why in the "Comments" column. Some areas always worry me: external interfaces, itty-bitty details of screen behaviour, agreeing things with clients, acceptance testing, and support during initial live use, for example. Some undoubtedly reflect my own skills, or lack of them; for example comms is not my strongest area so interfacing always tends to get a big contingency.

I do not believe that contingency can be measured to a greater accuracy than 5%. As I said, I tend to use 15% as a minimum if the functionality of the system is known to a reasonable degree (more if it isn't, less if it has been frozen). I rarely find a figure of over 100% is needed; it is usually easier to cut out the task altogether and state "The estimate does not include X, Y and Z".

3.9 The Remaining Code/Test Tasks

We'll assume now that you have an estimate for the baseline task and a level of contingency for it. The next bit is a little easier because we can estimate the remaining code/test tasks in terms of their difficulty relative to this.

What I do not condone is the method commonly seen of dividing tasks into "easy", "medium" and "hard" and making the same estimate for each type. This is lazy and produces enormous inaccuracies. Typically the "easy" task is over-estimated, leading to lists of similar routines being allocated (say) 3 days each. Alternatively some real ballstranglers are listed as "difficult" and given a single (usually underestimated) number of days – such as 12. If you know enough about the tasks to classify them then you ought to know enough to give a considered evaluation to each, based on your estimate for the "typically simple" task.

If you are an intuitive sort of person then there is no mystique in my recommended method. Suppose we said our baseline task took 10 days with a contingency of 25% then we might look at another task and think something like "There's a few more fields on the screen, maybe another couple of entities to read from the database and a bit of calculation so ... call it 15 days. The algorithm for the calculation isn't specified but it can't possibly be *too* complicated so we'll push the contingency for this task to 35%". Alternately you may say "The matching algorithm has no similarity to

the baseline task, it must be 6 or 8 times as complicated as that – so we'll call it 75 days with a contingency of 25%".

If you are not an intuitive type you'll have to do it the hard way. But before we think what this might involve let's look at the software tools available to the estimid.

4 Function Point Analysis and COCOMO

4.1 Science or Mysticism?

At every stage in my 'method' (if that is what it is) I am telling you to draw up lists of tasks, assumptions, rules and weightings which are *specific to your project*. And this can only be done by gaining as much information as you can about the tasks which make it up and then using the brain's natural abilities for analysis and synthesis to allow you to use your intuition, your knowledge, your experience, together with a rational approach, so that you feel comfortable with your estimates.

By contrast, function point analysis is a wonderful exercise for the dull-witted. There are lots of forms, little bits of detail, hours of painstaking work and, at the end, a number. A wonderful, mystic, magical number that has been divined by a method involving no imagination, no guesswork and the minimum of involvement with the individuality of the system. But is such a measure complete or accurate? Are these factors and weights relevant to Logica projects?

Everything about function points seems to me to be based on mainframes and files and inputs and outputs and all the rest of the stuff I threw out with my flowcharting template. It is certainly not appropriate for anything involving lots of calculation, real-time processing or communications.

But anyway, this part is harmless. A measure, any measure, of function size and complexity may have its uses. The hard part is converting this into a measure of the time it will take to implement the function, or even into lines of code. Here there are any number of formulae but even less science.

One example is the COCOMO tool, much touted within Logica. While can understand the motivation of those who enjoy a consultation with the cauldron on the blasted heath, I do not understand the use of a method which is far more difficult to employ than the simple task-by-task educated guess which I advocate. For in COCOMO you need to estimate all sorts of difficult things. Like the "programmer capability"– come on now, are they in the 15th percentile or the 90th percentile? A mixture? Sorry, COCOMO can't handle that. Or the "requirements volatility" – well, yes, just precisely how much will the requirements change as the project develops? I wish I knew. But in COCOMO you've *got* to say. And are its 16 cost drivers really the *only* things which affect how long a project will take?

As with function point analysis, I can't say I have much faith in the weighting factors. For instance, under the 'reliability' driver a "slight inconvenience" if a fault develops is rated as multiplying the effort by 0.75, while any "risk to human life" will multiply it by just 1.40. A human life finally valued – at less than twice a slight inconvenience. Tell that to NASA. And these factors are multiplicative, so by a judicious manipulation of the cost drivers you can alter the time taken for your project between 0.08 and 117 times the 'base' value. A base value which, let's remember, has more than likely come from that other wet digit, function point analysis. A guess multiplied by 16 guesses; and this is supposed to be a scientific method.

Frankly, if I knew how may lines of code would end up in a system – *really* knew as opposed to having an estimate from function point analysis – then the estimates would be easy. The list of cost drivers in COCOMO is a useful guide; but I'll make up my own list and weighting factors, thank you.

The temptation with COCOMO is to bump all the cost drivers up to the max under the "COCOMO says it can't take longer than this even if everything goes wrong" principle. The Business Manager chops off or adds on a bit and that is what ends up bid. A very satisfactory situation all round; everybody thinks they've worked hard on the estimate and it's nobody's fault if we lose out to a rival or if we win and it all takes twice as long. Naked and unashamed estimidity in action.

For the beauty of these methods is that you have only turned the handle; and if the estimates turn out to be thickly-sliced baloney with extra pickle then it's not your fault, it's the methodology. One sunny day, when we put the experience gained from projects back into the model, it will be perfect. Not in my lifetime, I suspect. In the immortal words of Sam Goldwyn, include me out.

A friend of Einstein once noticed a horseshoe hanging in the great man's lab. "Surely", he asked, "you don't believe in such pseudo-scientific nonsense?". "No", replied Einstein, "I don't. But they say it works even if you don't believe in it". And so it is with function point analysis and COCOMO. Nobody really believes the numbers they comes up with (do they?) but they still keep running it "just as a cross check". Well why not? That's what this paper is all about: making you feel good about your own estimates. You know the answer really; the hard part is justifying it to yourself and then steeling yourself to write it down and present it to someone else.

Another friend of Einstein once asked him how he approached the knotty dilemmas of everyday life. “Oh”, he replied, “I usually just toss a coin...”. “What?”, replied his questioner, “You – the great exponent of the scientific method – just toss a coin?”. “No”, said Einstein, “Let me finish. I toss the coin and then I ask myself *if I am happy with the result*”.

So, tot up some function points and after you have determined the number of lines of code ask yourself what this estimate feels like. For instance if the method comes up with approximately 14,256.6 lines of code ask yourself how many *pages* this would be (with headings, comments and so on). Imagine this pile of listings on the table. Do you really think the whole project can be done in this much code. Do you really think it will take *as much* as this pile? Are you happy with the result? And if not, what are you going to trust – the methodology or your intuition?

4.2 Do-It-Yourself Function Point Analysis

So no, I don’t think you should rush up to your nearest PC and start using whatever estimation tools some journal-reading fools have installed for the estimid. However, don’t let’s throw the baby out with the bathwater; if you have estimated your baseline task as I suggest and have got a good list of assumptions then why not use a sort of Do-It-Yourself approach whereby all the factors and weights are assigned by you?

Well, my simple answer would be that it takes too long. However, I do acknowledge that there are people who blanch at the thought of just producing numbers from thin air and demand some sort of mumbo-jumbo on paper to justify the result to themselves. It is very much like writing down the a long division sum when you know full well that you could do the arithmetic mentally.

So here is how you could go about formalising the method of converting your estimate for the ‘baseline’ task into estimates for every remaining task.

a) Identify the variables which will make any of the tasks more or less difficult than your baseline one (if you have more than one type of baseline task then you’re going to have to repeat this process for each one).

b) Decide whether these variables are part of the task or part of the contingency.

c) For the 'task' variables weight each of these using an arbitrary points system, making the baseline task equal 100.

d) For the 'contingency' variables estimate a range of percentage increases that will need to be added on or taken off if the specified variable does or does not occur.

e) Calculate the points total for all the other tasks relative to the baseline task.

f) You have already estimated the number of days the baseline task will take, so you can now put a number of days by all the other tasks.

g) Calculate the contingency percentage commensurate with the 'contingency' variables for each task and add it on.

h) Sit back and admire results.

Now at this point you probably would rather go back and join the group who can call figures from the vasty deeps. For it seems that there are actually more things to guess at using this method. Indeed this is so, but each of these stages is relatively easy to attempt (break it into little bits, remember) and you will have the satisfaction of a 'methodology' that at least looks scientific even if every stage of it is guesswork.

So let's try an example and assume our baseline task has an estimate of 8 days with a contingency of 25%.

a) Firstly we must identify the variables – here's a typical list:

(i) Number of fields on the screen.
(ii) Number of these fields that are inputtable.
(iii) Number of fields that need some calculation before output.
(iv) Difficulty of this calculation.
(v) Amount of calculation that is needed when the user hits 'enter'.
(vi) How well the algorithm for the calculations is known.
(vii) Approximately how many data entities we'll need to read/write.
(viii) How good a mental picture we've got of the data structures/ entities that will be involved.
(ix) Are some lookup tables going to be accessed – if so how many?

(x) Is there any flashy screen manipulation to be done?
(xi) Are there any complexities about the screen navigation to/from this function?
(xii) How many error messages may be issued.
(xiii) Is there any tricky validation or cross-checking involved?
(xiv) Is this function going to be particularly difficult to test for some reason?
(xv) How well do we understand how this function integrates with the rest of the system?

But of course the list you draw up will be specific to your project, and draw on your knowledge of the system, what documents you already have and so on.

b) Now we determine which variables affect the time taken to code/test the task and which affect the contingency. Looking at the list above it would seem that points (vi), (viii) and (xv) are to do with the contingency and the remainder is to do with the code.

c) The next step is to take the 'task' variables and give them a points weighting. The quickest and most intuitive way is to award exactly 100 points to the baseline task and then divide these among the variables. Easy. It also helps if you drop all the unimportant variables and just add on a few points if needed for a particular function

d) So I may decide on a points weighting for my particular project of:

20 Number of inputtable fields
30 Amount/difficulty of calculation
30 How many data entities we'll need to read/write
10 Amount of flashy screen manipulation to be done
10 Difficulty of testing

e) For the 'contingency' variables I am going to say:

5% - 20%	For not understanding the calculations too well
10% - 60%	For not having thought out all the data structures
10% - 70%	For not understanding how this function integrates with the rest of the system

Incidentally these are additions to the contingency percentage for the baseline task, not percentages of it.

f) So now we approach the estimation of each task. Lets say that the first one relative to the baseline task has:

- Twice as many inputtable fields.
- Has no more calculation.
- Will need twice as many entities to be read/written.
- Has no particularly flashy screen stuff.
- Will be about 50% more difficult to test.

This leads to a points score of 155 (40 + 30 + 60 + 10 + 15).

g) We estimated 8 days for the baseline task so this one will be 8 * 155 / 100 = 12.5 to the nearest half day (hell, this method isn't that accurate).

h) Turning to the contingency

- The calculations are about the same as the baseline
- We have a fairly good grasp of how we want the data to look in this area
- But we are a little vague (but not totally baffled) about where or why this function is used

So the additional contingency is 0% + 0% + (say) 10%. Adding this to our baseline contingency of 25% we get 35%.

i) And there we are – task 1 has an estimate of 12.5 days and a contingency of 35%. Now repeat for the remaining 165.

Now this may all look complicated and (worse still) subjective. But I would contend that the individual steps are fairly easy *so long as you have a good grasp of the overall project*. It is a methodology designed for the estimid because you end up with lots of individual little figures (why, you could even put them on a spreadsheet) and in turn this will give you a warm glow of faith in your figures. However, unlike 'real' function point analysis every factor and weight has been assigned by you.

However, it must be said that I would never use this method myself. I have needed to adopt it on one occasion because I was working with people who had no faith in intuition. Well, they may call it intuition: I call it experience. Even if I did use this method I'd still fix the results to align with my personal perception of the task. But then I believe estimation is an art, not a science.

5 The Project Time-Scale

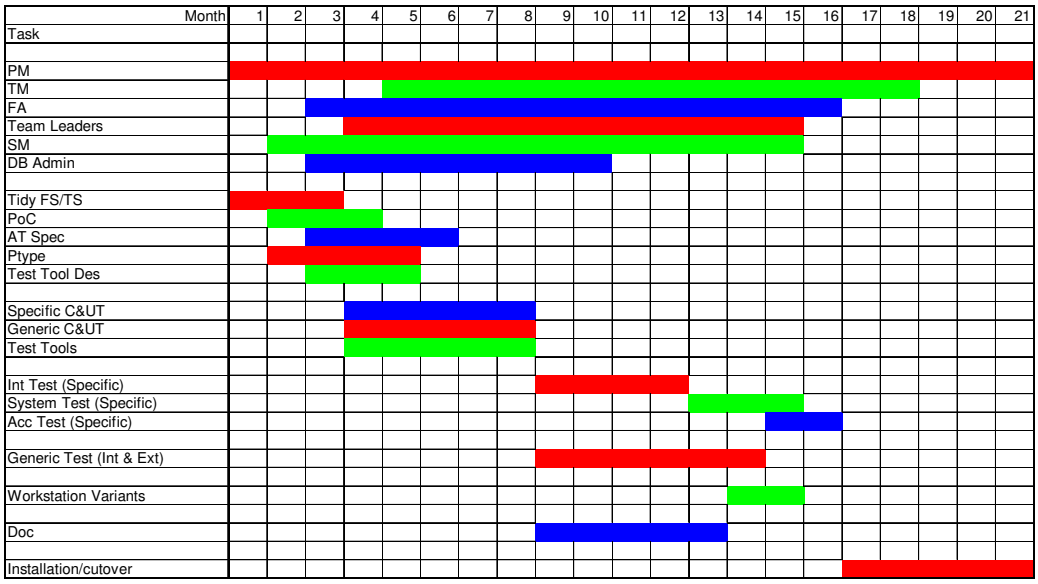
5.1 The Gantt Chart

Okay, by one means or another I will now assume you have estimated the code and unit test figures for each task, along with the appropriate contingency. You can now fill in your spreadsheet and calculate an overall code and unit test figure for the project. We can now turn our attention to the overheads. To do this it will be a great help to know how long each phase of the project will last and what the team-size looks like. Hence...

Pablo's Bloody Obvious Rule of Estimation Number 10

Sketch up a Gantt chart.

How you do this is up to you. You may prefer to use a drawing package but I have never found one friendly towards the large amount of amendments and corrections that need to be made. You may also be a fan of PMW or a similar package; it will come as no surprise to learn that I don't know how to use any of these, although I have sat next to people while they struggled and cursed in attempting to enter a simple modification. Hence I tend to either use pencil and paper, or Excel – as in the following:



Of course your chart will just be for the code/unit test bit at present; in the real example above I just have three code/unit test (C&UT) teams, who are covering “generic”, “specific” and “test tools” parts of the project.

In dividing the code/test work among a number of teams remember that:

- You shouldn’t have more than six in a team (including the team-leader).
- Team-leaders should not be assigned any code/unit test work unless the project is very small or the team is very small.
- Each team-member should have a continual diet of work throughout the project.
- There should be the minimum of dependencies between one team and another.
- All tasks must be complete within the overall code/unit test time available.
- Allow 18 days in a month.
- Allow for development team overheads (walkthroughs, reviews etc.). I usually allocate 0.5 days/week. This does not apply to team-leaders who are usually full-time overheads anyway.

You should use the ‘with contingency’ figures for the Gantt chart. By this I mean the individual task contingencies. At this stage I recommend ignoring the overall project contingency because it is more likely that this will be met by applying additional resources rather than extending the timescales.

However, on projects where the end-date is *really* fixed – space shots or general elections for example – a slab of naked and unadorned contingency just before that date might be seen as prudent.

5.2 Overhead Tasks

Let us turn then to the phases of the project other than code and unit test. Some, such as 'proof of concept' exercises are too specific to make general guidelines practical. However, there is one rule which applies to all documents, in particular Functional Specifications.

Pablo's Bloody Obvious Rule of Estimation Number 11

You can't write a document in less months than you have people.

If you have estimated 16 man-months to write your masterpiece then it *cannot* be done well in less than 4 elapsed months. Try, and you will end up with holes, inconsistencies, and a general level of quality which will take longer to put right than you saved by rushing the original task.

However, if the document really does split into separate pieces, or there is only a *small* amount of overlap, then a team may be allocated to each part and the rule applied to each. But you must be very sure that this assumption is correct; certainly Functional Specifications are a highly risky area.

5.2.1 Functional and Technical Specifications

In general, the time taken to construct the Functional Specification is proportional to the code and unit test time you have already allocated. The lazy way is to take a percentage of this (say, 20%) and then modify it according to your view of how difficult it is going to be to write the specification. One consideration here is whether to include screen prototypes; they may take a while to do but they are unambiguous, can be agreed with the potential users and sometimes may be recycled for production use. Remember that Functional Specifications have to be agreed with clients or users and this process is often iterative and time-consuming.

Technical Specifications are also dependent on the amount of code and unit test time and may be tackled in the same way as the Functional Specification. There is often a long initial period of discussion, design and

walkthroughs before the details of each routine can be written, so it may be preferable to estimate this separately.

All projects should have someone assigned to be the 'keeper of the vision', Functional Authority and general guru on what the system should do. This role, which may be full-time depending on the size of the project, involves processing change requests and clarifying the functionality of the system to the project team and (more usually) the client. It is a particularly important role during the construction of the Functional Specification.

Similarly, I would recommend a single person is assigned as Technical Authority, whose remit is to maintain the Technical Specification. All too often when new staff start on a project they are given this document (or the equivalent) and told that it "gives a good idea but is not up-to-date". This is inexcusable but acts as a reminder that time should always be allotted to keep *all* project documentation valid.

5.2.2 Testing

Testing is consistently under-estimated, for two reasons. Firstly there is a need to write a test specification. Obviously this needs to be created before the testing begins, which usually means that it must be kept in synch with changes to the code which are made during development. Failure to do so means that the testing is constantly stalled by disagreements between the testers and the developers. In the case of the Acceptance Test Specification, which is usually written at the start of the project, all changes must be agreed with the client. So you have documents which are very long and boring to write, and which must be kept up-to-date. All very time-consuming.

Secondly an allowance must be made for fixes to the code which arise as a result of testing. This is often forgotten or under-estimated.

As testing is largely proportional to the amount of code, and should decrease at each stage, you may like to enter this into the spreadsheet to be calculated automatically. For example

Total Code and Unit Test	1000	With Contingency
Integration Test Spec	100	10% of Total C&UT
Integration Test	150	Incl. bug fixes. 15% of Total C&UT
System Test Spec	80	8% of Total C&UT
System Test	100	Incl. Bug fixes. 10% of total C&UT
Acceptance Spec	50	5% of Total C&UT
Acceptance	80	8% of Total C&UT

Need I tell you that the percentages are dependent on your own project? Integration testing is dependent on the complexity of the software subcomponents and the amount of integration needed (e.g. with bought-in software). System testing is more dependent on the amount of functionality and the amount of interrelationship between the software subcomponents.

Acceptance itself always takes longer than you think because it is often the first time that some of the client staff have seen the system. This inevitably leads to what might be termed a training session under the guise of disputes about functionality. I often include a ‘dry run’ of the complete acceptance procedure so that the team is sure that there will be no problems.

5.2.3 User Guides and Help Text

Another task often forgotten is help text for the screens. This may include help for individual fields, messages, and even on-line documentation. Again this can be simply estimated, for example by taking the number of screens and estimating that, say, four can be done in a day. Check the tools available for help file creation and editing.

User Guides are another contentious issue. It is tempting to estimate a couple of weeks while reassuring the client that a fully-comprehensive guide will be supplied. Check what sort of guide the client might expect (given good on-screen help text for instance). In the worst case the guide will need to be properly laid out and printed, like the Microsoft documentation, but usually a good word-processed document will suffice. Again, this can be estimated by taking the number of documentable functions and estimating, say, half a day to describe the operation of each. Don’t forget things like operator’s instructions and explanations of messages.

Additional technical documentation should be a thing of the past. Up-to-date Technical Specifications and well-commented code should be sufficient in most cases – but make sure the client understands this.

5.2.4 Support and Warranty

These sort of tasks can be estimated as a number of people for a number of months (or weeks). For example we may say 4 people to help out the users and solve small problems for the first two months, followed by 1 person for the next year. Or, more likely, we may say we are offering 90 days warranty so allow one person 1 day a week to provide support and solve bugs: total 18 days of the most junior person.

5.3 How Long Will the Project Last?

There are two possibilities here:

1. The end-date has already been determined (e.g. the client has stated that the system must be up and running by a certain date). This makes life easier (for now!) because you have figures for the number of days for each task, you know the dependencies between them (e.g. you can't start coding until the specifications are complete), and can split the available time up *pro rata*. Then you can see how many people are needed for each task. At this point you might decide that the project just can't be done in the time; for example PBOROE 11 might be broken or the coding phase needs more people than the computer industry currently employs. Prepare your arguments, neaten up your Gantt chart and prepare to brave the storm that will greet your announcement that the desired end-date cannot be met.

2. The end-date is up to you. This is a little more difficult as you need to decide on a logical division of work among the teams. Often there is a logical split, for instance between the 'generic' and 'specific' parts of the project in my example Gantt chart. Bear in mind, of course, that someone is bound to say "can't it be done any quicker?" so you need to think about what sort of timescale will make the whole thing impossible. If the pressure is on for a quicker delivery then the 'logical' plan will have to be abandoned. It probably would never have happened that way anyhow.

So now your spreadsheet should have all the overhead tasks, other than the management overhead, filled in and these should be reflected in your Gantt chart.

6 Project Roles

It is time to approach the problem from the other direction and apply:

> Pablo's Bloody Obvious Rule of Estimation Number 12
>
> Make a list of project roles.

All project roles require different mixtures of skills and experience. It is important that we assign the right levels of staff to each task so that the estimate is correct in money terms. We could either treat the estimate as an academic exercise and price a hypothetical, ideally-qualified person to each logical role and then price the job from that, or we could assume continuity of staff between the different phases of the project, presupposing a set of ideal staff each with the right mixture of skills. The first is often easier, although possibly less accurate; the second should be used if a significant number of the team-members have already been nominated.

Some possible roles to consider are:

- Project Manager
- 'Keeper of the Vision'/ Functional Authority
- Technical Manager
- QA
- Database design experts
- Test database maintenance
- System management
- Team leaders
- Experienced analyst programmers
- Other analyst/programmers
- Consultants with specific knowledge
- Performance analysts
- Prototype developers
- Programmers
- User interface experts
- Screen designers
- Test team leaders
- System/Integration/Acceptance Test team

- Test spec writers
- User manual writers

For each role you must ask yourself:

- How many people are needed.
- When they will be needed.
- How long they will be needed for.
- Who they are going to report to.
- Could they be used at any earlier or later points in the project (e.g. can Technical Spec writer No. 3 become Team Leader No. 4?).

Obviously, all this all needs considerable thought. To have an extra person on the project for the duration can push up the price considerably. Conversely to forget that someone is needed can eat into the project profits very quickly. As an example, I have for many years just assigned one junior person (or less) to system management tasks, having a mental picture of backups and the occasional call to the manufacturer's service engineers. However, projects are now often done using lots more third party software, for instance for networking, database, configuration management and 4GL. The poor old system manager is then considered responsible to the smooth operation of all this; a considerable task, particularly at the start of the project. So think about all these roles in depth, and, need I say it, relate them to the particular needs and quirks of the project in hand.

Consideration of the project roles should enable you to sub-divide or re-draw your Gantt chart to show the assignment of each individual to each task. The times allowed for all the project overheads can also now be added into the chart. The result may be something like this:

Year			1996								1997								Phase 1	Phase 2	Total
Month	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Days	Days	Days
Role/Team																					
Project Manager	1	1	1	1	1	1	1	1	1	1	1								108	90	198
Development Team Leader	1	1	1	1	1	1	1	1	1	1	1								108	90	198
Functional Authority	1	1	1	1	1	1	1												54	72	126
Analysts	2	2	1	3	5	5	4	1											90	324	414
Technical Architect			1	1	1	1	1	1											36	72	108
Technical Designers			1	1	2	3	3	4											36	216	252
System Management	1	1	1	1	1	1	1	2	2	1	1	2							126	144	270
Forte Consultant		0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25							27	22.5	49.5
Senior Developers			1	1	2	2	2	2	2	2	2								144	144	288
Developers				1	1	1	2	2	3	4	4								90	234	324
Development Support								1	1		2	1							36	54	90
Senior Test Designer						1	1	1			1	1	1						54	54	108
System/Integration Testers						1	1	1			2	3	4						54	162	216
User Acceptance Testers							1	1	1				2	4	4				54	180	234
Database Design & Maint		1	1	1	1	1	1	1	1	1	1								108	72	180
Support Co-ordinator									1	1	1				1	1	1	1	54	72	126
Support & User Training									1						2	4	4	2	18	216	234
BSF Infrastructure	1	1	1	1	1	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25				99	36	135
Team Profile	**7**	**8.25**	**10.25**	**13.25**	**17.25**	**19.5**	**20.5**	**19.5**	**14.5**	**11.5**	**16.5**	**7.5**	**7.25**	**4.25**	**7.25**	**5**	**5**	**3**	**1296**	**2254.5**	**3550.5**

In this example, the project is in two phases. For each role the number of people needed is entered into the Excel cell, allowing the overall team profile to be seen at the bottom. The number of days for each phase is calculated separately and totalled. These totals can be compared with the task-by-task estimate as a cross-check.

Inevitably there will be some ‘slack time’ for some team members. One view on this is that the roles and timescale are purely hypothetical at this stage and hence the slack time should not be included in the estimate. A more pragmatic view is that people do not enter and leave projects at the desired times and tend to hang on for the duration, hence this time *should* be included. I side with the first view on the grounds that in practice no project has ever proceeded in accordance with a plan drawn up at the estimation stage. It is unlikely that people will be idle; they will either be doing estimated work or additional work which should have been covered by the overall contingency. Hence the slack time should *not* be included.

You should now also be able to build up a diagram of the team structure at each phase in the project. This may consist of ‘logical’ roles, several of which may be filled by one actual person.

7 Project-Wide Contingency

7.1 Risk Analysis

Earlier we identified our project-wide assumptions and defined them as *risks*. A typical risk is "TP Monitor will introduce an unacceptable performance overhead". The correct approach to these is to produce a Risk Analysis which discusses each one under the following headings:

Risk A brief description

Actions What steps are being/can be undertaken to mitigate the risk

Impact Number of days or percentage increase to the estimated project time that will be incurred if the identified risk actually happens

For example:

Risk Object-oriented design when not all designated staff have completed an OO development of a similar size.

Actions Try to get as many experienced OO people involved as possible. Training may help a bit.

Impact Assume 20 inexperienced staff and a delay of 1 month each to get things sorted. Total 360 days.

There are some risks which apply to nearly all projects:

- A general under-estimate.
- Requirements not sufficiently tied down.
- Unrealistic timescales.
- Doubts about bought-in software.
- Doubts about the architectural solution.
- Performance and sizing constraints.
- Staff experience or capability.
- Untried development methodology.
- Poor contact with end users.
- Inadequate development environment or tools.

The end result of the Risk Analysis will be a number of days which represents the overall project contingency, to be added on to your total man-

day figure in the spreadsheet. As for the task contingency, you should be prepared to reduce this if someone can set your mind at rest as to the assumptions you have made.

7.2 Client Staff

There is an increasing trend for joint projects, which tosses yet another unknown into the estimation stew. At one extreme the business knowledge of the client staff, and improved relations generally, may result in less work and less risk. At the other extreme the client may put in inexperienced people, with the aim of training them up, resulting in a distinctly negative effect on progress despite the nominal addition of manpower.

My inclination in these circumstances is not to assume any change from an estimate which assumes an all-Logica team. However, the use of client staff must be considered as a risk. It may not be politic to present it to the client as such but they must be made aware that the supply of ineffective staff will cause delay. The contingency for the risk may include bringing ‘x’ client staff up to speed, plus and allowance for replacing ‘y’ client staff who prove completely ineffective.

7.3 Allowances for Reuse

Pressure often arises from salesmen or clients to reduce estimates in order to take account of what they see as positive factors. These include use of client staff, reuse of existing code, similarities between different parts of the system, reuse of knowledge and expertise from other projects, and use of packaged solutions. Sometimes such factors are undeniably true (or the pressure to see them as such is severe) and an allowance must be made. This should be done when drawing up the task-by-task estimates, not by applying some blanket rule such as “reduce by 25% to account for reuse of code from the XYZ project”.

Usually, however, there is something of a grey area. Will these factors reduce the development time or not? After all, it can take longer to understand and make use of existing code, however well it is documented, than it does to write the actual code you need. Will these two routines, which seem so similar now, actually turn out to be so in 9 months time? And will we actually get these so-called experts from the ABC project which is due to finish just as this one starts?

If I do make an allowance for such things then my inclination is to assume the reverse as a risk. So I may accumulate 100 days off the code estimates to cater for reuse of some existing software, but then add all or most of it back as contingency to cover the risk of this code being inappropriate. If possible, therefore, I prefer to let it all cancel out and ignore the positive factors. If there eventually proves to be a benefit then the project gains some time – but this is bound to disappear through some other unforeseen circumstance.

8 Estimating the Project Cost

This should be simple to derive from your Gantt chart. Get the rates for each person or role, enter these as a new column in the spreadsheet, multiply up, total the lot and, hey presto!, an estimated price will emerge. An example spreadsheet is shown at the end of this section.

For Logica staff and contract people, this process is easy because the man-day rates (Logica UCR) are known. Clients have more of a problem because they usually have not calculated how much each staff member really costs. It is not just a question of salary but also of support staff, buildings, equipment costs and so on. This is a major reason why Logica's rates (or, indeed those of any software suppler) always appear so large; clients have no realistic internal costs to compare them against. It also leaves a serious problem where Logica is trying to provide an overall cost for a joint development. Usually the best we can do is quote a number of person-days.

If you don't have the actual names of nominated project staff then fill in an 'ideal' person who you know could do the job. It doesn't matter if they are on site in Bangalore and unlikely to be free until doomsday. Just say to yourself we need a 'Fred Smith' sort of person here and a 'Bill Jones' sort of person there. As the actual names are added, keep an eye on the rates and skills of the staff involved; cheaper (or unknown) staff may cut down the cost but maybe the estimate or contingency should be upped. Conversely it is wrong to assume that better-grade staff will do a faster job. They may do a better one (you may care to reduce the contingency) but coding speed is not a function of seniority.

Remember too, that if the project crosses a year boundary then you will need to guess at next year's rates.

Much of all this is handled automatically on Logica's QMS form which, in any case is an essential part of an internal Bid Review. However, I don't intend to cover the QMS in this Guide.

The following is the 'prices' part of a spreadsheet for a three-phase mixed-team project, with the different costs and contingencies for each phase being shown. Man-day rates were difficult here as some roles were specifically assigned to the client, some specifically to Logica, and the remaining man-days divided equally. Hence the need for a 'unified rate' which combines the Logica rates with a flat £400 a day which the client wished to use to cost their own staff.

Month / Role/Team	L Rate (£) Discounted	Unified Rate (£)	Rate Comments		Cost P1	Cost P2	Cost P3	Total Cost
Project Manager	790	400	Client		43200	36000	165600	244800
Development Team Leader	790	790	Logica Senior PM		85320	71100	327060	483480
Functional Authority	690	690	Assume V.C.		37260	49680	347760	434700
Analysts	490	445	Logica Intermed. BA/Client		40050	144180	400500	584730
Technical Architect	690	690	Logica Senior Consultant		24840	49680	335340	409860
Technical Designers	490	445	Logica Intermed. TA/Client		16020	96120	344430	456570
System Management	490	400	Client		50400	57600	165600	273600
Forte Consultant	1600	1600	M.O.		43200	36000	165600	244800
Senior Developers	690	545	Logica Senior Cons./Client		78480	78480	313920	470880
Developers	250	325	Logica Intermed. Imp./Client		29250	76050	549900	655200
Development Support	250	325	Logica Intermed. Imp./Client		11700	17550	128700	157950
Senior Test Designer	310	355	Logica Senior Test/Client		19170	19170	83070	121410
System/Integration Testers	200	300	Logica Junior Test/Client		16200	48600	329400	394200
User Acceptance Testers	200	300	Logica Junior Test/Client		16200	54000	151200	221400
Database Design & Maint	690	690	Logica Senior TA		74520	49680	260820	385020
Support Co-ordinator	310	355	Logica Senior Test		19170	25560	70290	115020
Support & User Training	200	300	Logica Junior Imp/Client		5400	64800	226800	297000
BSF Infrastructure	690	690	R.O.		68310	24840	0	93150
Team Profile				**Total (£)**	**678690**	**999090**	**4365990**	**6043770**
		400	Flat Client Rate	Contingency %	10%	20%	20%	
				Contingency £	67869	199818	873198	1140885
				With Cont.	**746559**	**1198908**	**5239188**	**7184655**

9 Reviews

9.1 Tidying Up

At this point you should clean up your documentation. As always, presentation is important and gives your estimate the look of authority. At this stage you should have:

1. A spreadsheet with all the tasks and overheads, plus an estimate and contingency for each one, with task specific assumptions listed.

2. A spreadsheet for the Do-It-Yourself Function Point Analysis, if you chose to use this method.

3. A Risk Analysis resulting in a total project-wide contingency.

4. A Gantt chart (which, hopefully, is now in synch with the spreadsheets).

5. A team structure for each stage of the project.

6. A profile of the team size at each month during the project.

7. A list of roles and rates, giving a project cost (ex. expenses, hardware etc.).

Right or wrong, at least it will look like you have given some thought to your estimate.

9.2 Types of Review

Some sort of review of your estimate is essential – not to spread the blame and reduce your estimidity, but just to get someone to point out the tasks, concerns or assumptions which you have left out. So...

Pablo's Bloody Obvious Rule of Estimation Number 13

Have someone else review your estimate.

If a review cannot be done for some reason then this is a new risk and should be added to the project contingency.

There are several approaches to the review process:

9.2.1 Peer review of an existing estimate

The snag here is that the reviewer usually does not have the in-depth knowledge of the original estimator. This gives the reviewer a panic attack of estimidity and he then suggests that all the estimates are bumped up a bit. The original estimator is then only too glad of an excuse to do so, knowing that he won't have to justify the increase and BANG! The project never starts.

A worse scenario is where the estimator sticks grimly to his figures and the reviewer then is left as a Cassandra, muttering "I think it's all hopelessly underestimated but no-one will listen".

To avoid these situations such a review should be treated as a chance for the reviewer to raise queries and points that the original estimator may need to consider. This gives a chance for the estimator to take these factors on board if they wish, without the ability to shift the responsibility on to the reviewer. If the reviewer still has serious doubts about the figures then they can be raised with management. However, the person with the in-depth knowledge should really take the responsibility here.

9.2.2 Team Estimate from a List of Tasks

This also has its problems. In my experience there are two scenarios

- One person in the team dominates the effort and takes the key decisions (e.g. estimation of the baseline task). The rest of the team then breath a sigh of relief and fill in the easy bits. In times of trouble it is then all the fault of the pathfinder and everyone else can claim to have raised doubts at the time which were not heeded. The benefit, if there is one, of teamwork is thus negated.

- Everyone is too estimid to venture any opinions. The result is that anyone who is foolish enough to name a figure is immediately picked on by the rest of the team who want it increased. Needless to say, the eventual estimate spirals to be so over-cautious that it is useless.

I don't believe estimates can be done as a team unless it is by a...

9.2.3 Combined Walkthrough and Estimate

Each task is described by someone and then two or more people write down their figure. These are then compared, discussed and agreed. The process allows concerns to be raised and fully discussed in context. These are often factors not previously considered at all, which shows one of the advantages of this method. The process does not suffer the disadvantages of the team review because the function under consideration is so fresh in everyone's mind and no-one has a previous view to defend.

In my experience, this method works very well. I have even undertaken a joint client/Logica estimate in this way. The only snag is that the exercise can take several tiring days.

9.2.4 Two Separate Estimates

Obviously the crunch point comes when the two are compared. This can have the advantage of showing up tasks, concerns or assumptions that were forgotten by the other person, particularly true if the two people use different methods of estimation. One useful technique is to agree the task list first, then go away and undertake the separate estimates.

The snag arises when there are widely differing views on the effort required for a particular task. The coward's way out is to take the highest each time. Usually one estimate is consistently higher than another so the 'loser' naturally feels his work has been in vain. An even worse scenario can arise when some decision-maker decides to take the lowest from each set of figures out of expediency. All this must be fought. If the concept of doing two separate estimates is to be of use then the comparison must be thought out on a task-by-task basis, possibly monitored by a third party. Again, it is usually the case that one of the estimators has a greater depth of knowledge of the situation than the other, and should ultimately take the responsibility.

9.3 Now Do the Project

Once the estimate has been reviewed and revised then there is nothing more to be done. In Logica the next stage is the Bid Review, during which some assumptions may be questioned and the contingency altered. Rarely, in my experience, are the raw estimates affected.

Assuming the project goes ahead, there is nothing more to do than sit back and acknowledge the plaudits of your colleagues, as they undertake a well-planned and well-estimated set of tasks, and your superiors, as they appreciate the precision with which they were able to plan and control its costs.

10 The Post Mortem

The only thing we learn from history is that we learn nothing from history. A project goes bad, loses millions, causes sweat and heartbreak for the team and at the end everyone breathes a sigh of relief – and gets stuck into the estimates for the next one. What are its chances? Well, not too good really. No-one will want to repeat their recent unpleasant experiences so the estimates are going to be cautious, really cautious, cautious enough so the project never starts, in fact. And as for that idiot who did the estimates – well we'd better make sure he gets nowhere near the estimation for this one...

This is the wrong use of some valuable experience. You are a better estimator after some failures, not a worse. The way to exploit that experience is to be constructive. And the way to do that is to analyse the estimates by means of a post mortem. So this is...

Pablo's Bloody Obvious Rule of Estimation Number 14

Hold a post mortem.

Of course this should be done for projects that go to plan as well as those which go badly; there is just as much experience to be gleaned there. The reasons a post mortem is so rarely held are

- The project changed out of all recognition during the development, so making the original estimates redundant.
- It would be difficult without pointing the blame at individuals.
- Everyone is fed up with the bloody thing.

Of these the first is the most relevant. The main reason projects fail is, of course, badly managed changes to the specifications. But why were they badly managed? Could the changes have been foreseen? These are the sort of questions you should be asking in your post mortem. Here is a short check-list of the sort of things that should be covered:

- Did the project meet the timescales/budget, if not by how much either way?
- Was the estimated team structure maintained – if not, why?
- Was the list of tasks in the original estimate correct? If not what was added/removed and why?
- Were the assumptions met? If not was the contingency sufficient to cover them?
- Were the 'raw' estimates for each task correct. If not, why?

It is possible to answer these questions without personal acrimony so long as the post mortem is kept within the team. It is not necessary to write anything down. Just get the facts and remember them. That way, your estimates, like Gary Player's putting, will continue to improve.

11 And Finally...

That's all there is to estimation. You may think, having read this paper, that you still don't know how to go about doing the estimates for your next project. You're right. There is no magic formula, no easy way, no algorithm, no archive of statistics[1], nothing that will make your task any easier. What I have hoped to do is to make you realise that you *do* have all the experience that you need, and that you *can* produce a set of figures that you feel good about. Nobody can ask more than that. And once you have succeeded you can at least have the personal satisfaction that many of your contemporaries would have shrunk away from such a task. But having done the best job you can, and maybe once the project has ground into action with your estimates and your plan firmly in place there may be moments when the full horror of what you have done sweeps over you in a tidal wave of apprehension. So there is one universal rule which I have stolen...

Pablo's Bloody Obvious Rule of Estimation Number 15

Don't panic.

And happy estimating...

11.1 Further Information

Please feel free to contact me if you want any further advice or help with your estimates.

Paul Coombs, Logica UK Ltd, 75 Hampstead Road, London NW1 2PL
Tel: 0171 637 9111
Email: coombsp@logica.com

The views expressed in this Guide are those of the author and should not be interpreted as Logica's policy or methodology in any way.

---

[1] Yet, although a Process Improvement Team under the Logica Cortex initiative is working on this,